Final Report
ARPA Phase I SBIR Project ARPA 94-083
Contract DAAH01-94-CR257

A SIMULATION QUERY LANGUAGE FOR THE NEXT GENERATION OF DISCRETE EVENT SIMULATION SOFTWARE

August 21, 1994 - March 20, 1995

Network Dynamics, Inc.
128 Wheeler Road
Burlington, MA 01803
617-270-4120
617-270-4119 (fax)

95-J-1919

I. PROJECT SUMMARY & REVIEW OF OPPORTUNITY

Simulation is widely recognized as one of the most versatile and general-purpose tools available today for modeling complex processes and solving problems in design, performance evaluation, decision making, and planning[1]. Examples where simulation is frequently used include C^3I environments, manufacturing systems, and computer networks. In fact, most situations that confront decision makers today (from financial planners to designers of highly sophisticated engineering systems) are of such complexity that their analysis and solution far surpass the scope of available analytical and numerical methods; this leaves simulation as the only alternative of "universal" applicability.

The importance of simulation has given rise to a number of commercially available software packages (e.g., SIMAN, SLAM, SIMULA, SIMSCRIPT, MODSIM) whose applicability ranges from very generic to highly specialized. However, the use of typical simulation software is limited by the following factors:

1. One must have thorough knowledge of the specific tool at a detailed technical level before attempting to use it in a modeling effort.
2. One must be an experienced programmer, in addition to a decision maker.
3. In order to make decisions based on simulation, one usually needs to run multiple simulations and then carefully manage all output data collected on a case-by-case basis
4. The field of simulation was developed primarily as a special branch of statistics involving dynamical phenomena. There is less emphasis on the computer science aspects of the subject[2]. Consequently, manual handling and analysis of input/output data is still the norm. Design of interfaces, component ware interoperability, intelligent and automated analysis of output have been neglected. For example, the practice of Object Oriented Programming (OOP) [1], with few exceptions, is still nascent in simulation languages despite the fact that the OOP idea actually originated in simulation. Lastly, hardware advances, such as massively parallel computers and workstation networking, are only beginning to be noticed in simulation theory and practice.

---

[1] U.S. Department of Commerce, *Emerging Technologies: A Survey of Technical and Economic Opportunities,* Spring 1990
U. S. Department of Defense, *Critical Technologies Plan* , 15 March 1990
Council on competitiveness, *Gaining New Ground: Technology Priorities for America's Future,* 1990
Office of Science and Technology Policy: *22 Technology Critical to Economic Prosperity and National Security,* April 25, 1991

[2] A quick search of the Proceedings of the 1994 Winter Simulation Conference reveals only 7-16% (depending on definition) of the papers presented deal with CS aspects of the simulation.

A-1

5. The ultimate purpose of simulation is often system performance evaluation and optimization. However, simulation is notoriously computer time consuming when it comes to parametric studies of system performance. Unless substantial speedup of the performance evaluation process can be achieved, systematic performance studies of most real-world problems are beyond reach even with supercomputers.

As stated in the description of the ARPA 94-083 solicitation topic, there is a parallel between databases and simulation in the following sense. Query languages (such as SQL [2]) are intended to make the intricacies of computer databases transparent to casual users; similarly, a "simulation query language" should play that role with computer simulation. The challenge in this case is much greater, however, because a simulation is a dynamic object: the state of a database is affected by external inputs which add/delete entries, whereas the state of a simulation continuously changes as a result of the internal dynamics of the process being simulated. For example, a snapshot of a database conveys all the information contained in that database; a snapshot of a simulation may not convey any information at all if one is not aware of the specifics of the process being simulated, how long it has been since the simulation started, or (in some cases) the statistical significance of the simulation output data being queried at any given point in time.

We believe that there is a real opportunity to develop a high level "simulation query language" or "system performance evaluator" for a broad application domain that encompasses all *Discrete-Event Dynamic Systems* (DEDS). DEDS are characterized by dynamics governed by *events*: The state of the system can only change as particular events take place over time. Some events are controlled, while others are uncontrolled (spontaneous) and often random in nature. For example, a C^3I environment is a discrete event dynamic system, where controlled events can be message receptions, actions, or commands, while uncontrolled events may be various equipment failures. A manufacturing system is another example, where events correspond to the beginning or end of a machine processing cycle, causing changes to various inventories. Making decisions and planning production in this setting is a typical situation where simulation is used. Financial planning and processing is also a setting where discrete-event simulation finds many applications. In this case, events such as "purchase", "make payment", or "collect revenue" drive changes in various budgets or financial sheets. In fact, DEDS are pervasive in modern technology largely because of the use of computers which, in themselves, are discrete-event systems.

In what follows, we have tried, whenever possible, to avoid needless duplication of text with the Phase II proposal that is being submitted concurrently with this Phase I final report. Some amount of overlap, however, is unavoidable.

II. DESCRIPTION OF PHASE I RESULTS

We begin by reviewing the original objectives of the ARPA 94-083 Phase I SBIR project:

- A Simulation Modeling Framework for Discrete Event Dynamic Systems (DEDS)
- A Query Language with Emphasis on Highly Interactive Features
- A Feasibility Demonstration of the Above.

Shortly after the award of the contract for Phase I we learned of the existing I^3 (Integrated Intelligent Information) effort at ARPA and the related KQML language. This I^3 effort is a visionary design of computing environment where different computing applications such as, databases, expert systems, etc. as well as simulation models can interact with each other directly without manual handling and intervention. The operative word is **interoperability** and the attendant technology is **component ware** with the KQML (Knowledge-based Query and Manipulation Language) as the common language for communication to achieve seamless integration.

In addition, new theoretical advances in the area of parallel simulation and performance optimization since the time of writing of the original Phase I proposal led us to the following accomplishments in our Phase I work which represent a re-definition and targeting of our original Phase I goals.

1. Learning about KQML. Since it is the expectation of ARPA that the proposed query language for discrete event simulation will be a component of the KQML environment and be able to interact with other agents, it became clear that we need to become conversant in agent-oriented programming and become part of the KQML community. This we have done.
2. Demonstration of the Efficiency of Concurrent and Parallel Simulation. The basic tenet here is that in the simultaneous simulation execution of a set of parametrically different but structurally similar systems, much computational load can be shared to increase efficiency. This increase can be achieved even on a traditional sequential computer. In this case, we refer to the methods developed as *concurrent* or *multithread* simulation, since, in effect, simulation runs corresponding to several systems are concurrently produced. Depending on hardware, as well as on the simulation software environment used, speedup (when compared to brute force repetition of the simulation experiments) can be of the order of 10 or more. In a parallel processing environment the speedup increase is spectacular. In this case, we refer to the methods used as *parallel* simulation, to distinguish them from concurrent simulation. We investigated the many ways this efficiency increase can be carried out and demonstrated its

feasibility by implementing the idea on a simple queuing system as well as on a more complicated semiconductor manufacturing simulation model.

3. Demonstration of one Type of High-Level Simulation Query. We took the viewpoint that the ultimate goal of simulation is performance evaluation, and, if possible, optimization. As such, the systematic exploration of the performance surface as a function of design parameters is a basic problem. The fact preventing this goal from being realized is the time-consuming nature of the simulation process. Recent theoretical advances in stochastic optimization, culminating in the *ordinal optimization* approach, however, promise to speed up this process by orders of magnitude. Furthermore this approach seems to be ideally wedded to the high level queries we visualized earlier. Consequently, we implemented and validated this combination of ideas via a software demonstration of the problem of quickly narrowing down the search for optimum in system simulations.

What our effort in Phase I taught us is the conviction that simulation can really be used very effectively in the exploratory design and operational tuning of discrete event systems. In other words, we have demonstrated the feasibility of successfully overcoming the fundamental limitations 4-5 mentioned in Section I. Difficulties 1-3 will also greatly lessen with the full-scale implementation of what we will propose. When properly integrated into the component ware environment (e.g., using NQML), a high-level simulation query language can fulfill the promise for which simulation was originally intended.

II.1 UNDERSTANDING THE ROLE OF SIMULATION IN KNOWLEDGE BASED SYSTEMS

The long-term objective of this effort is to integrate simulation into Knowledge Based Systems (KBS). When an end user runs an application, this application often involves retrieving information from a real-world system (through various sensors) or from some database. A simulator is a replacement for a real-world system when the latter does not actually exist. Alternatively, it provides the mechanism for testing the behavior of an existing real-world system under hypothetical conditions without actually implementing them. For instance, before implementing a complex plan that involves the mobilization of expensive resources over some period of time, one wishes to explore a multitude of scenario so as to differentiate "good" from "bad" plans and possibly identify an optimal plan.

Note that the output of a simulator consists of data normally placed in a database. The interaction between a simulator and such a database, however, is highly dynamic. As an example, if a user requests information from a specific database and this data are not found, the responsibility of the KBS may be to:

- locate the appropriate simulator which can generate the requested data
- provide the input necessary for the simulator to operate
- determine if the simulation output data are reliable (e.g., are they statistically significant?)
- organize the output data in a way that can be entered into the database in an appropriate form

This process is illustrated in **Fig. 1**. Currently, the functionality described above is performed purely manually. That is, the KBS has no capability to interpret user requests so as to locate the right simulation tool or databases, nor can it act as an intelligent coordinator between the two.

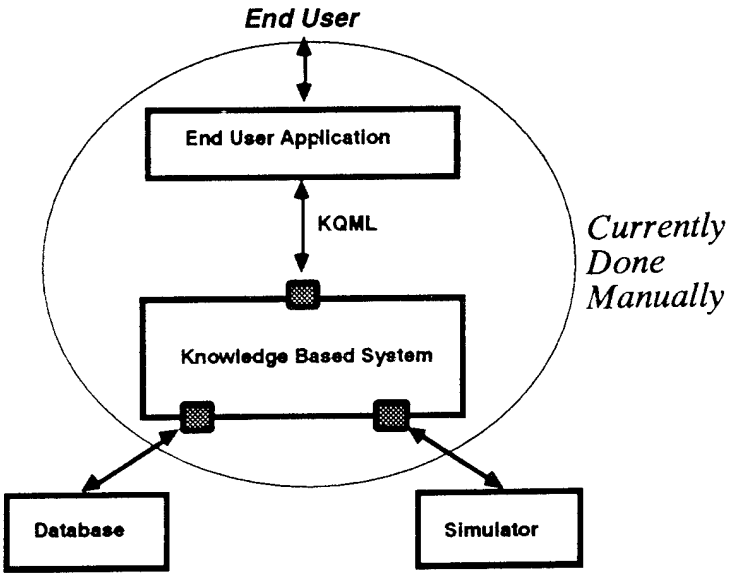

Fig. 1: Integration of Simulation into Knowledge Based Systems

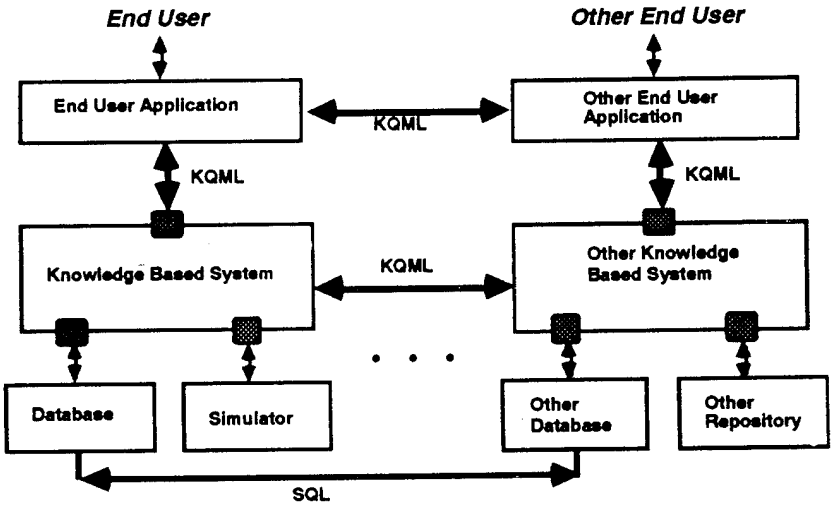

Fig. 2: Interoperability among agents in Knowledge Based Systems

In **Fig. 2**, a high-level architecture is shown, illustrating the role of KQML (Knowledge-based Query & Manipulation Language) as a language and protocol to support *interoperability* among *agents* in distributed applications.

The purpose of a *simulation query language* is to establish a systematic way to carry out (automate) this process as much as possible. The highly dynamic nature of the simulation/database interaction described above is responsible for the complexity involved in attaining this goal. As shown in **Fig. 3**, a simple query generally involves multiple simulation runs and there are at least two types of questions that the query processor must be able to handle: (a) Are the simulation output data statistically meaningful?, (b) Have enough simulation runs been performed to provide adequate information for responding to a given query?

In **Fig. 4**, we offer a specific example to illustrate these points. The query here requires a simulator to test a total of R options, one for each number of resources. It is possible that at least part of such information already resides in a database. In general, however, that is not the case. If R is extremely large, then it may only be feasible to run a number $n < R$ of simulations within a reasonable time frame. The first difficulty is to ensure that each of these n runs is statistically meaningful. This normally means that a longer run may be required until some acceptable level of confidence is attained. At the end of this stage, one typically obtains information which can be graphically represented as shown in the first plot in **Fig. 3**: a plot of time required to perform the given task as a function of the number of resources involved. Note, however, that only a rough approximation of the overall performance curve is obtained in this manner. In this example, one can focus on the range of resources whose performance is close to the given target T, say $[a, b]$. Thus, at the next stage of the query processing one needs to perform additional simulation over this range, in order to determine the best number of resources (the value x shown in the second plot in this figure) which constitutes the response to the query.

As a result of this effort, our Phase I objectives were refined to

- Developing high-level commands and queries for simulation modeling and performance evaluation of DEDS
- Speeding up simulation experimentation and performance evaluation and optimization
- Enhancing our understanding of KQML and relationships to existing query languages such as SQL.

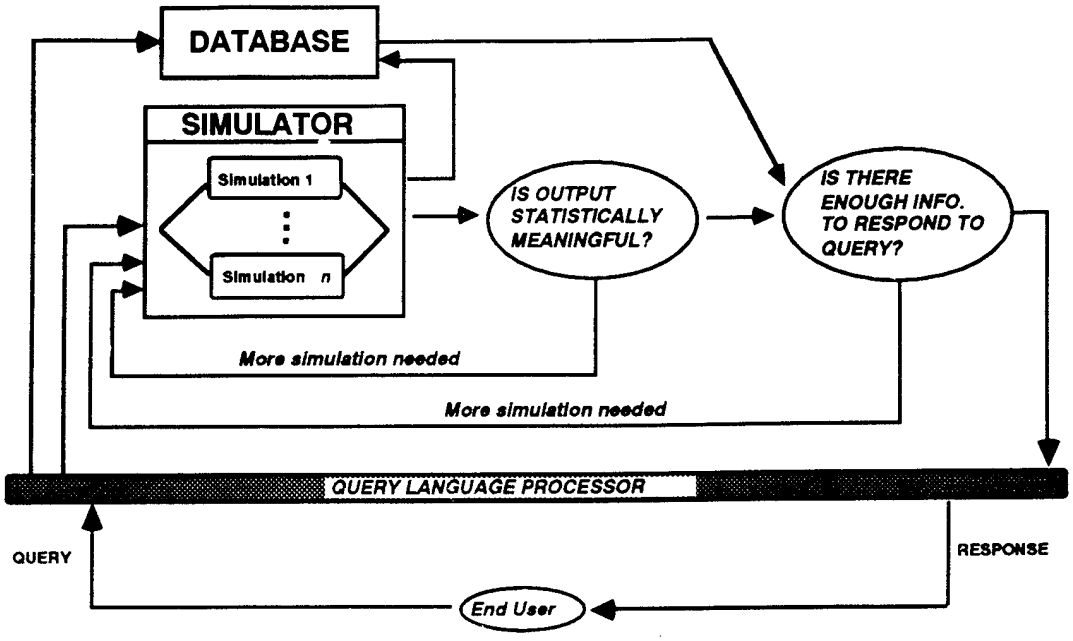

Fig. 3: Simulation Query Processing

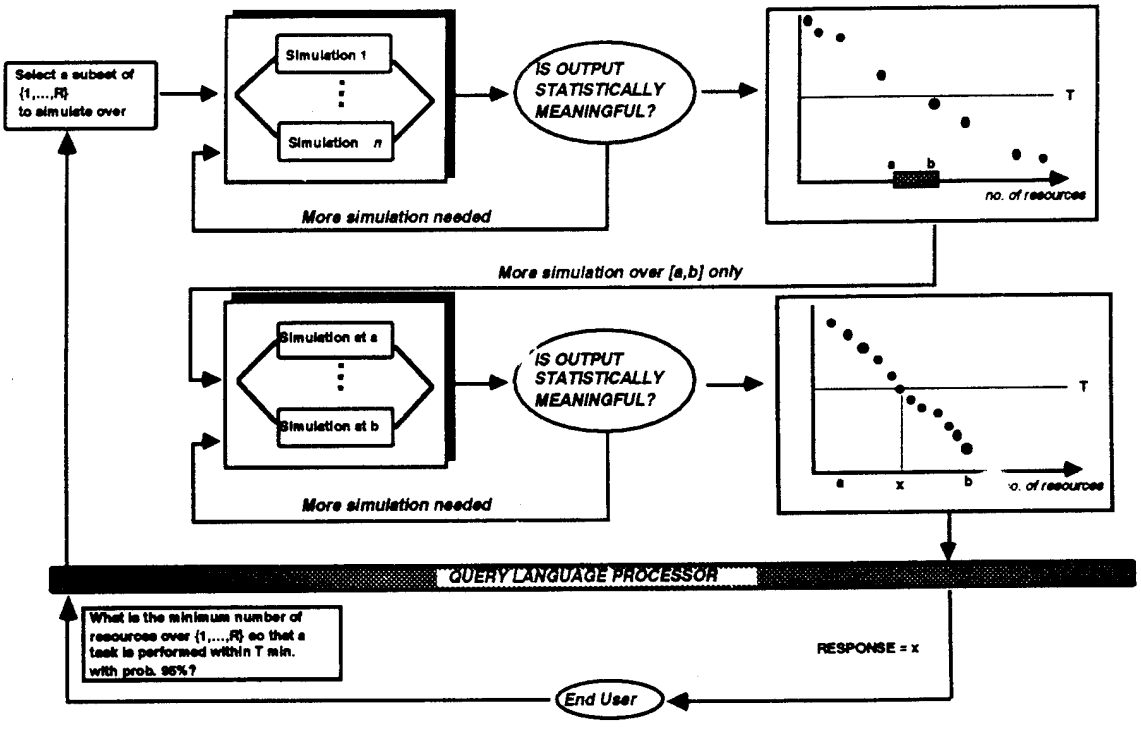

Fig. 4: Example of Simulation Query Processing

II.2 DEMONSTRATION OF SPEEDING UP SIMULATION VIA CONCURRENCY AND PARALLELIZATION

The basic motivation for this objective is the time consuming nature of system performance exploration: to obtain answers to N "what if..." questions, $(N+1)$ simulations are needed. Therefore, our goal is the following: *From a **single** simulation, obtain answers to all N "what if..." questions simultaneously.*

The method we have developed is based on using all possible data involved in one simulation to drive N simulations in parallel. This is illustrated in **Fig. 5**, where it is assumed that a model of a DEDS is given and a set of parameters (actions, designs) $\Theta = \{\theta_1,\ldots,\theta_m\}$ is specified. The problem then is to observe the DEDS under θ_1 and from the observations made to estimate/predict/infer/learn the DEDS performance under all $\theta_2,\ldots,\theta_m$ on line and in parallel.

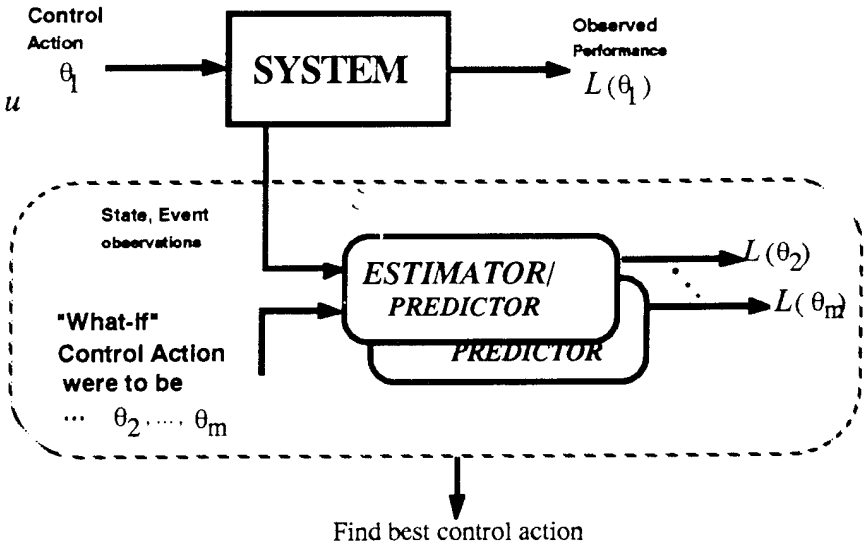

Fig. 5: Parallel simulation framework

In discussing parallel simulation, it is important to distinguish between the following different views:

1. BRUTE-FORCE REPETITIVE SIMULATION: simulate a system N times sequentially
2. PARALLEL SIMULATION OF **ONE** SYSTEM: distribute the simulation code over N processors or network of N workstations
3. PARALLEL SIMULATION OF **MANY** SYSTEMS WITH THE SAME STRUCTURE, BUT DIFFERENT SETTINGS: distribute simulation experiments over N processors, all driven by the same input data

A more appropriate term to use is *multithread* simulation:

- Set up a single simulation
- Generate multiple simultaneous *simulation threads*
- Each thread corresponds to a different system (same *structure*, but different *design*)

When implementing such an approach on ***sequential*** computers, processing is serial, but all simulation threads unfold in parallel. The main benefit here lies in sharing of data and of some data structure maintenance. On ***parallel*** computers, on the other hand, one processor is allocated to each simulation thread. In this case, the additional benefit is that all system state updates are also parallelized. To differentiate between these two implementations, we refer to the former as ***concurrent*** simulation and to the latter as ***parallel*** simulation.

There are two specific methods we have investigated for concurrent and parallel simulation: the Standard Clock (SC) approach and Augmented System Analysis (ASA). We will discuss these two approaches in the following two sections, and then proceed with specific numerical results to demonstrate the speedup effect of these methods compared to brute-force simulation.

II.2.1. Standard Clock (SC) Method.

In this approach we take a radical departure from traditional simulation approaches while at the same time making a rather restrictive assumption that all event lifetimes are exponentially distributed. However, this assumption can be overcome through various approximation techniques we have found to provide very good results. Assuming that every event $i \in E$ has exponentially distributed lifetimes, it suffices to associate with it a single parameter λ_i, the rate characterizing this distribution. It is then well known that when the system enters state x, the distribution of the ensuing interevent time is exponential with rate given by:

$$\Lambda(x) = \sum_{i \in \Gamma(x)} \lambda_i \qquad (1)$$

Moreover, when the state is x, the distribution of the triggering event is given by:

$$p(i, x) = \frac{\lambda_i}{\Lambda(x)}, \qquad i \in \Gamma(x) \qquad (2)$$

where $p(i, x)$ is the probability that the triggering event is i when the state is x. Another well known property of Markov chains is also exploited, known as uniformization: regardless of the actual event rate $\Lambda(x)$ for any state x in (1), we use a uniform rate given by

$$\Lambda = \sum_{i \in E} \lambda_i \qquad (3)$$

and treat all events which are not feasible in x and contributing the event rate $\Lambda - \Lambda(x)$ as "fictitious events". If such an event is observed in the simulation when the state is x, it is simply ignored.

Note that the mechanism through which the triggering event at any state is determined now becomes independent of the state. Specifically, in (12) we can now set $\Lambda(x) = \Lambda$ for all x. Therefore, the triggering event distribution at any state is simply $p_i = \lambda_i/\Lambda$.

The algorithm through which we may obtain N simulation runs in parallel using the SC method is presented below. The state of the jth simulation, $j = 1,\ldots,N$, is denoted by x^j and the corresponding total event rate (which may differ across experiments) by Λ_j.

Standard Clock Algorithm

Step 0. Initialize: x^j to a desired initial value for all $j = 1,\ldots,N$.

Repeat the following steps for all $j = 1,\ldots,N$:

Step 1. Generate an interevent time V with distribution $1 - e^{-t}$, $t > 0$.

Step 2. Generate an event type e with distribution $p_i = \lambda_i/\Lambda.$, $i \in E$.

Step 3. Check for event feasibility: if $e \in \Gamma(x^j)$, then go to *Step 4*, else skip it.

Step 4. Update state: $(x^j)' = f_j(x^j, e)$.

Step 5. Rescale interevent time: $V^j = V/\Lambda_j$ and return to *Step 1*.

The SC algorithm is also presented in **Fig. 6**, in order to contrast it to the conventional simulation approach. Observe that the determination of event times is totally shared by all simulations; in a SIMD parallel computer, the generation of interevent times may therefore be performed at the front-end computer. In addition, *Steps 2,3,5* above can be executed by a single instruction. In many cases, the parameter Λ_j is unchanged across simulations, so it is not needed; in this case, the distribution used in *Step 1* is simply $1 - e^{-\Lambda t}$, $t > 0$.

II.2.2. Augmented System Analysis.

Unlike the SC approach, in Augmented System Analysis (ASA) the parallel sample path construction process is driven by a specific simulation run already available. This presents the practical advantage of not having to redesign a simulation software environment to accommodate the SC setup described in the last section. To describe the ASA approach, let Σ_1 denote the simulation model already available and $\Sigma_2,\ldots,\Sigma_N$ the additional simulation models to be run. Let $\{x_k^j\}$ be the state sequence corresponding to Σ_j. Then, the evolution of the joint state (x_k^1, x_k^j), $k = 0,1,\ldots$, for any $j = 2,\ldots,N$ is viewed as the sample path of an "augmented system" which is driven by the exact same event sequence as the observed system Σ_1. As long as the observed state and the state of some Σ_j, $j = 2,\ldots,N$, satisfy the "observability condition" $\Gamma(x_k^j) \subseteq \Gamma(x_k^1)$, then every observed event in Σ_1 is simply used to update in parallel the state of Σ_j. Intuitively, the observability condition indicates that the observed simulation experiment contains at every state all the event information necessary to proceed with state updates. If, however, $\Gamma(x_k^j) \supset \Gamma(x_k^1)$ for some

state, then there is missing information and the simulation run corresponding to Σ_j must be "suspended" until some later time when the observability condition is again satisfied. The resulting procedure is known as *event matching*. It can be shown that this suspension preserves all statistical properties of the sample path being constructed in this fashion as long as all event lifetimes are exponentially distributed.

In order to describe the ASA event matching procedure more precisely, let us define a set $A = \{2,\ldots,N\}$, i.e., A contains the indices of the N-1 simulation experiments to be constructed in parallel. Let us also limit ourselves to parameters which do not affect event lifetime distributions; these are typically structural parameters such as the capacity of a queue or the population size of a closed queuing network. If following a state transition, $\Gamma(x_k^j) \supset \Gamma(x_k^1)$ for some $j \in A$, then j is removed from A until some future transition which causes $\Gamma(x_k^j) \subseteq \Gamma(x_k^1)$ to be satisfied; at this point, A is updated to include j once again. In other words, A represents the set of "active" simulation runs at any given time. In what follows we drop the subscript k and denote the current and next state of Σ_j by x^j and $(x^j)'$ respectively.

ASA - Event Matching Algorithm

Step 0. Initialize: $x^j = x^1$ for all $j = 1,\ldots,N$, and set $A = \{2,\ldots,N\}$.

Repeat the following steps for all $j = 2,\ldots,N$ with every event e observed in Σ_1:

Step 1. Update state: $(x^j)' = f_j(x^j, e)$ for all j such that $j \in A$

Step 2. Check for observability: if $\Gamma[(x^j)'] \supset \Gamma[(x^1)']$, then go to *Step 3*, else skip it and return to *Step 1*.

Step 3. Remove j from A and return to *Step 1*.

The ASA procedure is also shown in **Fig. 7**, in order to contrast it to the conventional simulation approach. In this case, it is assumed that a simulation is already being performed, so that the objective is to generate a number of additional simulation runs in parallel. Similar to the SC method, ASA is based on the assumption that event lifetimes are exponentially distributed (actually, at most one non-exponential event lifetime distribution is allowed). In addition, there are cases where this assumption is not required; specifically, in the case where the system of interest is such that $\Gamma(x) = E$ for all states. In general, however, ASA needs to be extended to arbitrary distributions, which is possible at the expense of additional overhead. This is a problem we have left for further study in the proposed Phase II project.

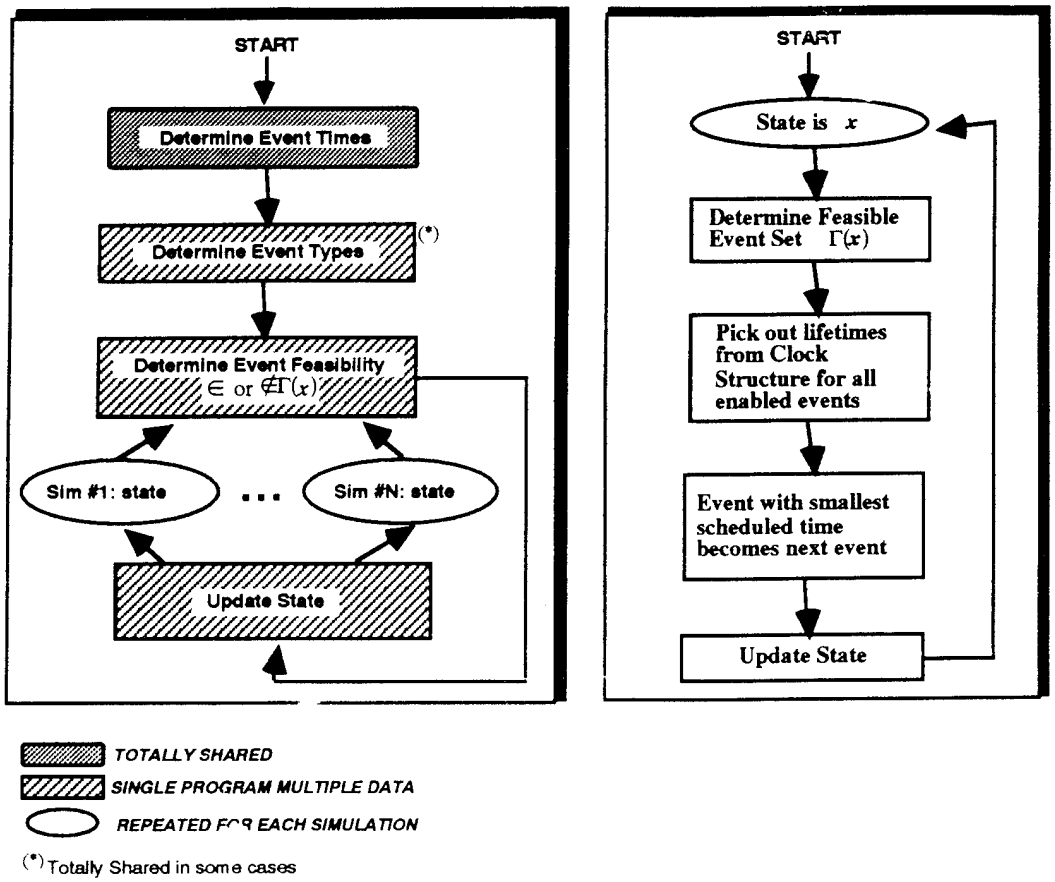

Fig. 6. The SC Simulation scheme compared to conventional simulation

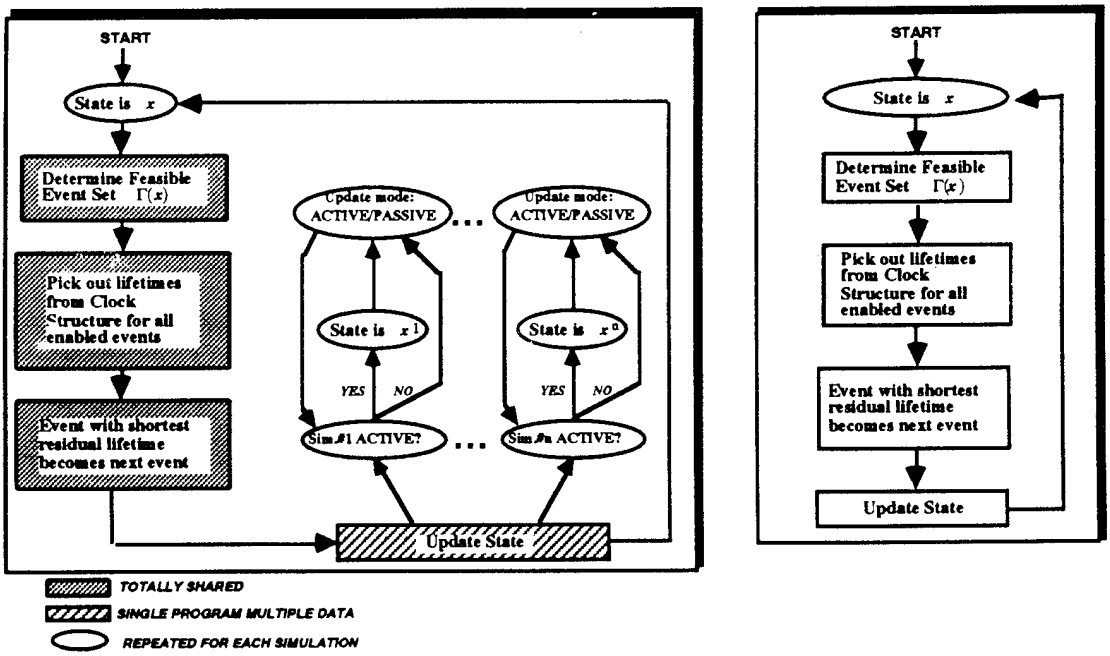

Fig. 7: Augmented System Analysis (ASA) vs. Conventional Simulation

II.2.3. Efficiency Study Results.

During the course of this project, we implemented both the SC and ASA approaches described above and applied them to various problems in order to evaluate their efficiency relative to the "brute-force" method which we have taken to be the obvious baseline approach. We have limited ourselves to sequential computers since access to a parallel machine was not feasible under the budgetary and time constraints of a Phase I project. However, it is easy to extrapolate to parallel computers, since one can only further benefit from a parallel processing environment. In this respect, the results reported in this section may be viewed as lower bounds to the speedup that can be achieved. We will limit ourselves here to two representative examples of the efficiency study conducted.

• **Example 1:** *Buffer capacity design for a simple queuing model.*

We consider a simple $M/M/1/K$ queuing system, where $K = 1,2,\ldots$ is the value of the buffer capacity where incoming customers are queued waiting to be processed. The system is simulated under $K = 2$ and we consider $K = 1,\ldots,10$ to be the range of parameter settings over which this system is to be simulated. Both the SC and ASA approaches were used and compared to "brute force" simulation (i.e., repeating the simulation 10 times for each of the 10 values of K). The results, in terms of total CPU time, are shown in **Fig. 7**. The CPU time required for each approach is plotted as a function of the number of parameter settings. The length of each simulation run was set so that the run stops after 100,000 total customer departures. In the SC method, recall that there are fictitious events generated, representing departures which are not actual ones; these are not counted as actual departures.

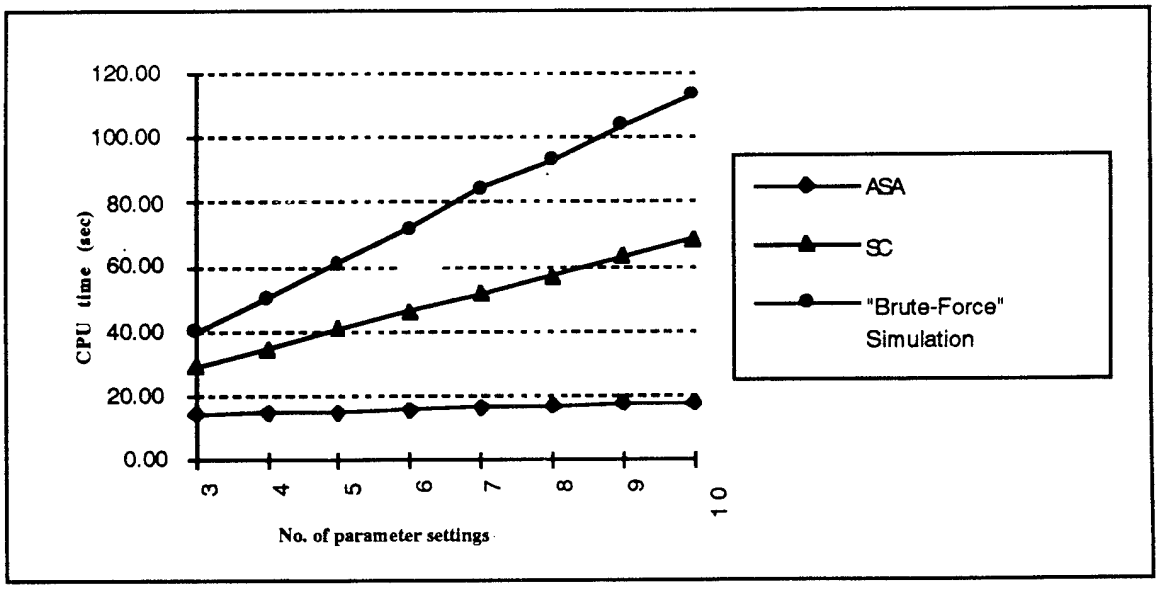

Fig. 8. Comparative results for Example 1

• **Example 2:** *Buffer allocation problem for a system of parallel queues.*

We consider a system of 6 $M/M/1/K$ queuing systems in parallel and a total of 80 buffer slots to be allocated among them. Customers arrive in a common stream and are routed to one of the queues according to some fixed probability distribution. In this case, we are interested in evaluating different buffer allocations of the form $(K_1,\ldots,K_6)$ where K_i is the capacity allocated to the ith queue. Clearly, there is an extremely large number of allocations to be evaluated. We limit ourselves to 10 randomly selected allocations over which we simulate the system. The stopping condition is such that at least 30,000 departures are observed in each of the 6 queues. Typical results (for one set of randomly selected allocations) are shown in **Fig. 9**.

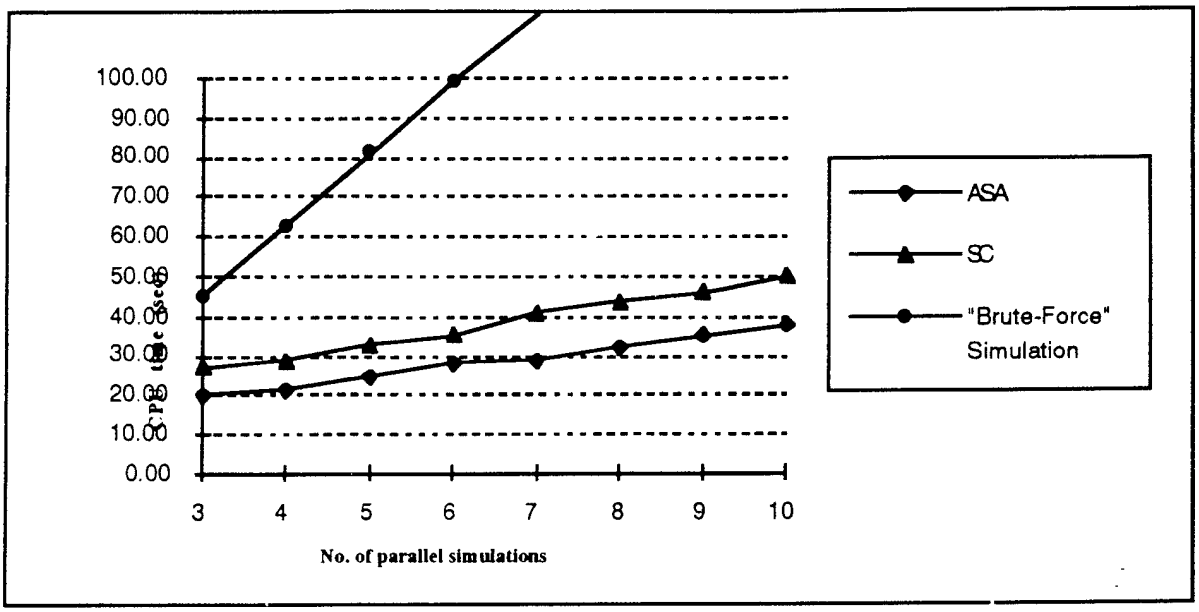

Fig. 9. Comparative results for Example 2

II.2.4. Measures of Efficiency in Concurrent and Parallel Simulation.

In order to evaluate the relative merit of different concurrent/parallel simulation schemes and obtain quantitative measures of their efficiency compared to brute-force simulation, it is important to define meaningful *efficiency metrics*. Our work during this Phase I project has led us to a number of possible such metrics; however, we believe that a substantial amount of further research is required to identify the most appropriate ones as parallel simulation techniques are further developed. We foresee this effort as part of a Phase II project.

We will limit ourselves here to one measure we have found useful in our study. We define the *speedup factor* for N parameter settings as

$$\text{SPEEDUP FACTOR}(N) = \frac{\text{Brute force simulation time for } N \text{ runs}}{\text{Parallel simulation time for } N \text{ runs}} \qquad (4)$$

Clearly, the speedup factor is dependent on N. However, as N becomes large, this number will converge to the following quantity:

$$\text{SPEEDUP FACTOR AS } N \to \infty = \frac{\text{Average slope of brute force simulation curve}}{\text{Average slope of parallel simulation curve}} \qquad (5)$$

which we can take as a first-cut measure of efficiency. As a representative example, we show in **Fig. 10** the speedup factor corresponding to the simulation time curves of **Fig. 11**. In the case of ASA, (4) gives an eventual speedup factor of 6.6; in the case of SC this is 5.5.

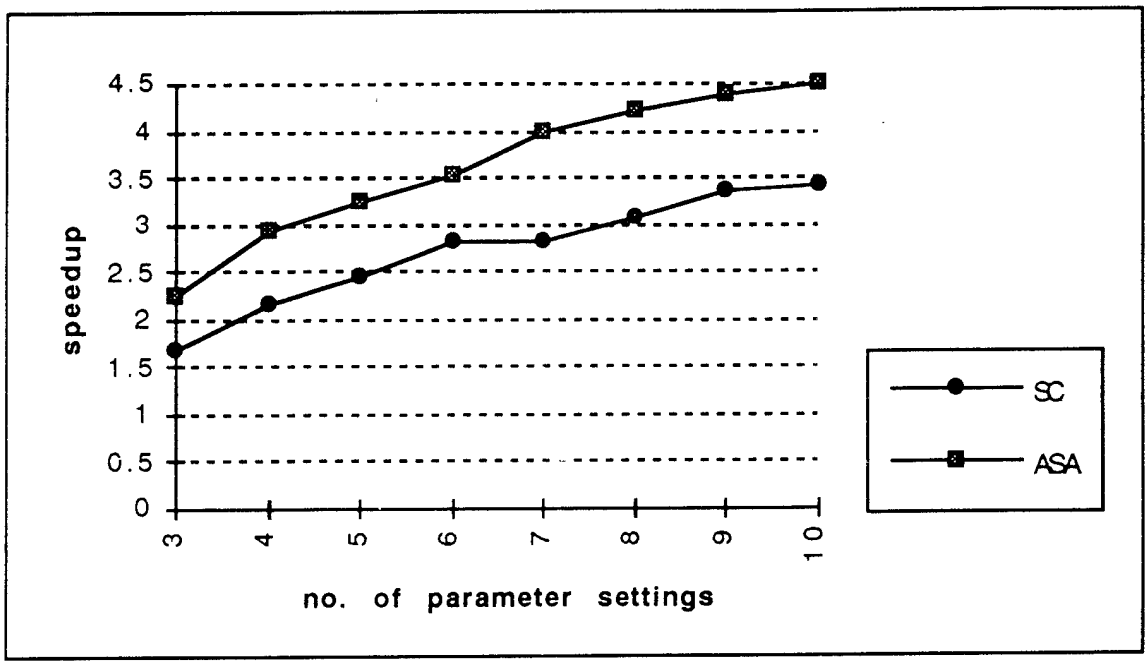

Fig. 10. Speedup factors for SC and ASA in Figure 8

We emphasize again that these results were obtained using our techniques on a *sequential* computer. If a *parallel* computer is used, the instructions marked "repeated for each simulation" in **Fig's 6-7** can be distributed over multiple processors. If the total time required for these instructions to be executed in a typical simulation experiment is T, then over N parameter settings the time required will become T instead of NT. This immediately results in a signif. ant additional increase in the speedup factors above.

II.2.5. Demonstrations.

The results shown in the previous sections were obtained using simulation code that we developed explicitly for the purpose of the efficiency study. In practice, however, it is obviously desirable to develop a parallel software simulation environment based on state-of-the-art software techniques (such as object-oriented programming) and simple to use user interfaces. We adopted the recent commercial simulation tool Extend in the Macintosh environment, and developed some simple

"objects", i.e., software modules, that can be easily added onto simulation models created through this tool. Our goal here was twofold: (a) Use the resulting software environment for proof-of-concept purposes, and (b) Perform an efficiency study similar to the one reported above in a commercial software simulation setting.

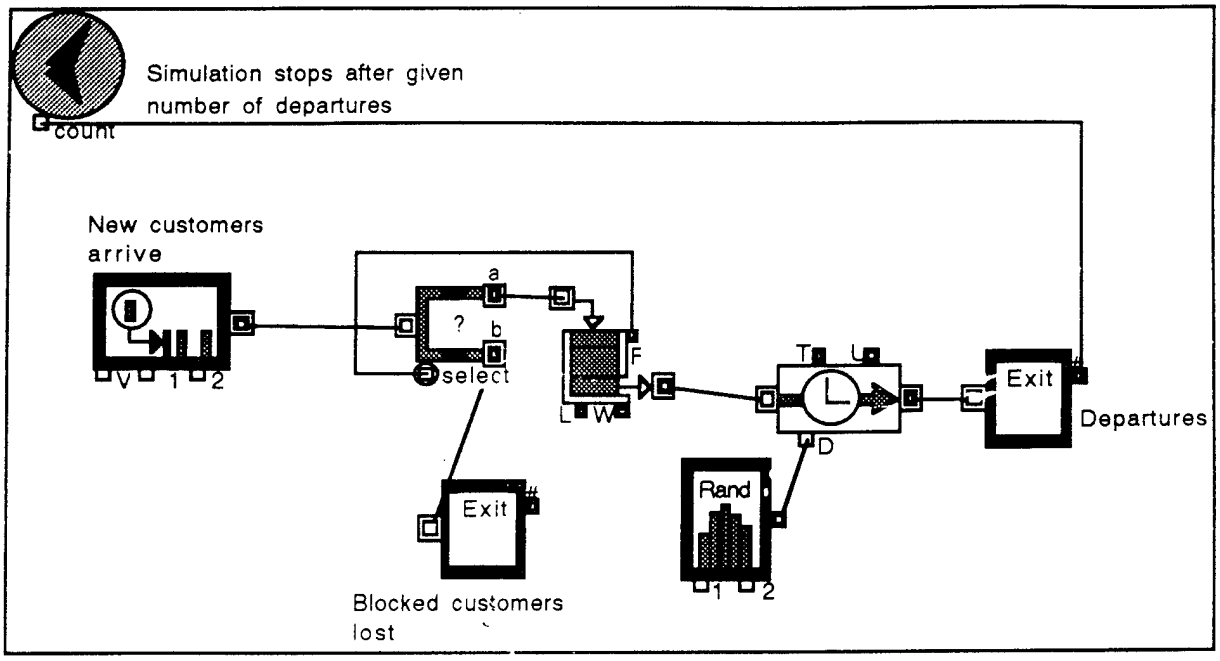

Fig. 11. Model of M/M/1/K queuing system using the 'Extend' simulation software

We will limit our results to the same simple model as Example 1 earlier, i.e., an *M/M/1/K* queuing model. The actual Extend model is shown in **Fig. 11**. In this model, there are two blocks used for random variate generation purposes: the one labeled "New customers arrive" and the one marked "Rand" which is used to assign service times to customers. The two blocks marked "Exit" are used to count the number of blocked customers (due to finding a full queue) and the number of departures. The simulation is set up to stop after a given number of departures.

In this effort, we considered only the ASA approach which is much easier to integrate into a conventional simulation tool such as Extend. In order to implement the ASA scheme, we created a block referred to as the "ASA block", and connected it to the model of **Fig. 11** as shown in **Fig. 12**. Specifically, the block was designed to sense every arrival and departure event from the actual model (in which a buffer capacity value is set) and evaluate the state of the system and the numbers of blocked customers and of departures for any desired number of systems with the buffer capacity value modified.

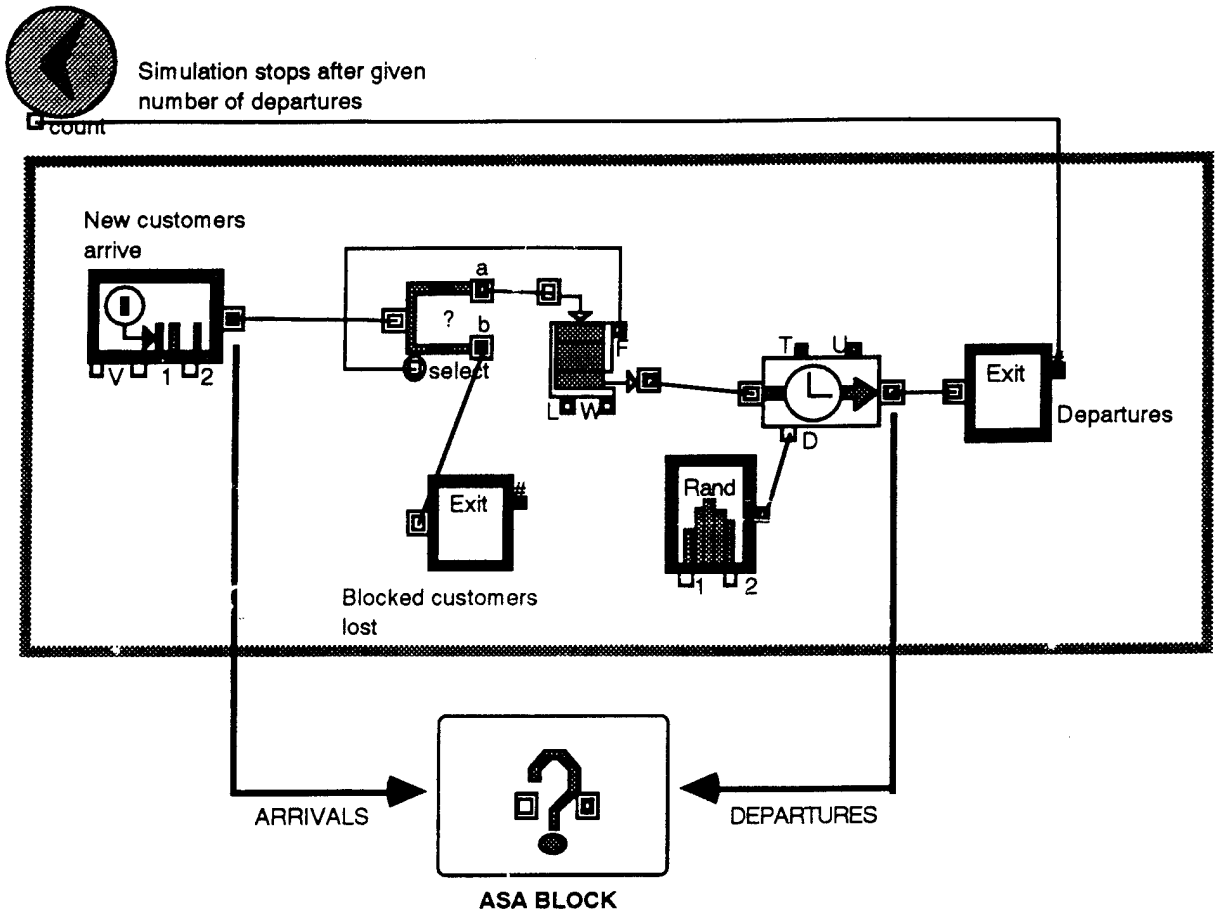

Fig. 12. Model of $M/M/1/K$ queuing system with ASA Block added for parallel simulation

Two different versions of the ASA block were implemented. In the first, referred to as "ASA1", the ASA procedure used takes advantage of the known structure of the actual system. This "customization" of the ASA block makes it particularly efficient. In the second version, the ASA block (refereed to as "ASA2") was not customized; instead, it was designed to process the arrival and departure events it sensed without any additional knowledge of the structure of the system. This obviously reduces its efficiency, but makes it a more general-purpose module one can insert into a large class of DEDS models simulated. A few representative results of this study are shown in **Fig. 13** (where the total CPU time for the brute force simulation approach and ASA1, ASA2 are shown) and **Fig. 14** (where the speedup factors of ASA1 and ASA2 are shown as a function of the number of parameter settings).

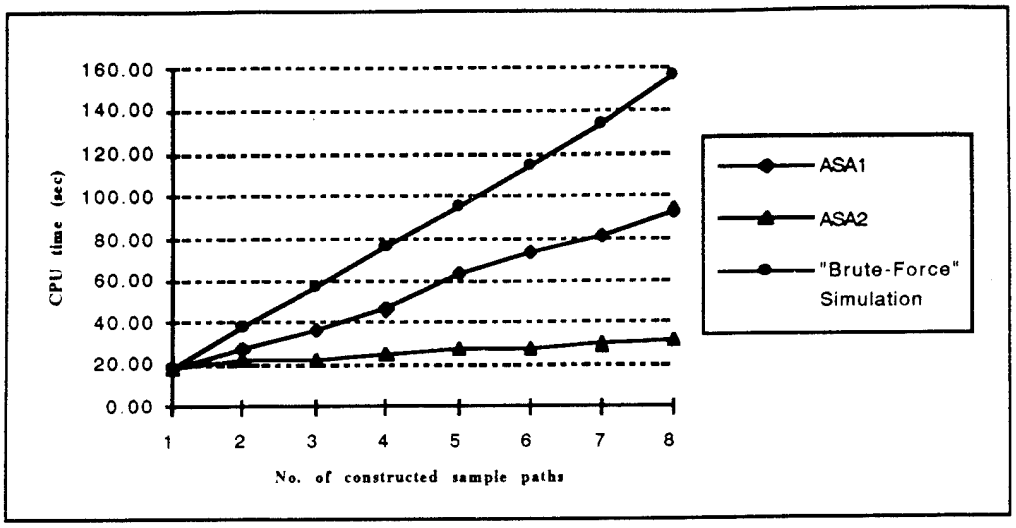

Fig. 13. *ASA Efficiency study results for the model of Figure 11*

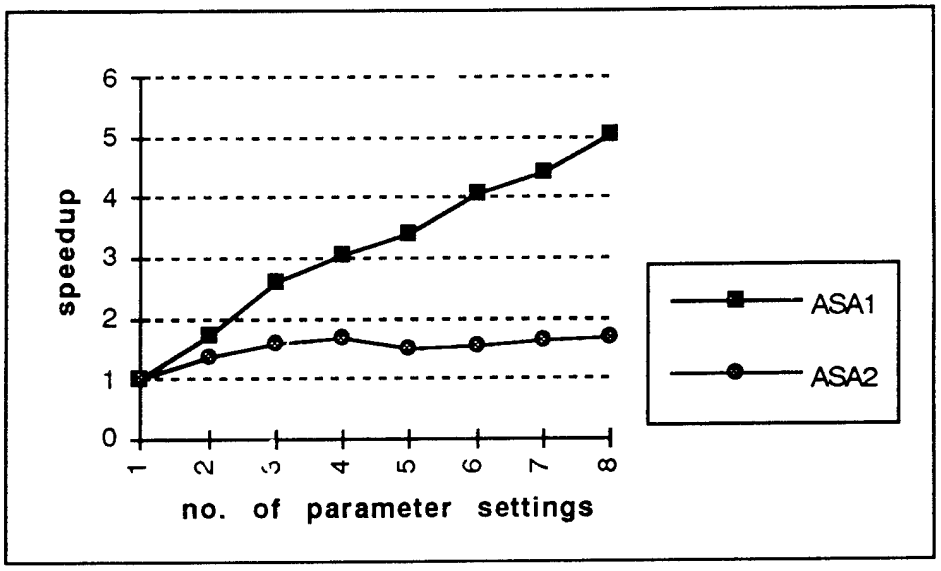

Fig. 14. *ASA Speedup factors for the model of Figure 11*

Lastly, we applied the ASA approach to a complex semiconductor manufacturing system (see **Fig. 15**), where the parameter of interest is taken to be the capacity of the queue shown. Note that the block marked OVEN is actually a hierarchical one containing a sub-model which in itself is quite complex. In this case, two modules were added to implement ASA.

The speedup results obtained for this model were comparable to the ones shown earlier. As already mentioned, since this method scales up easily, speedup in a parallel processing environment is essentially limited by the number of processors available.

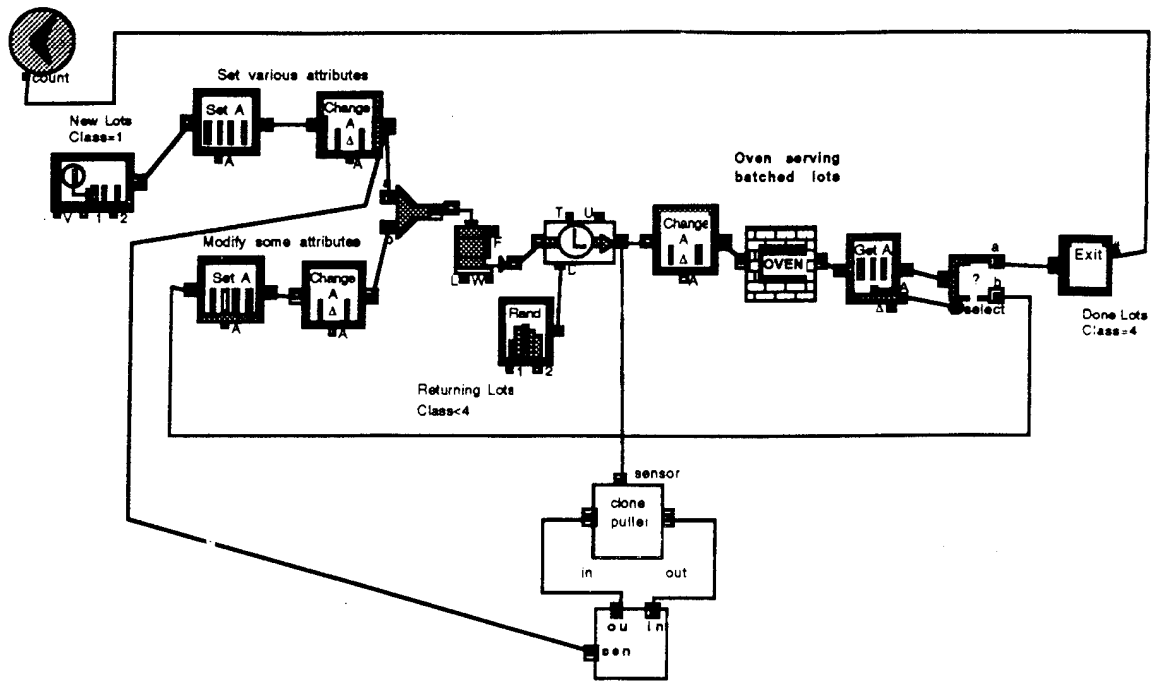

Fig. 15. Model of semiconductor manufacturing system with ASA capability

II.3 DEMONSTRATION OF INTELLIGENT EXPLORATION OF SYSTEM PERFORMANCE

The underlying rationale of this second demonstration is this: system performance exploration via simulation is most time consuming since each evaluation of system performance is a long Monte Carlo experiment. Exploratory search in the parameter space can be very costly in terms of both computing time and human labor. Thus, our goal is to quickly reduce the search region via the theory of ordinal optimization, i.e., locating the right ball park first. The methodology and the conceptual algorithmic steps are:

Method:

Use of a crude model to locate a subset of the search space that is *guaranteed* to contain "good designs" with high probability.

Steps:

- *Randomly* evaluate a set of performances using the crude model
- *Select the subset* of the top-*n*% of the crude performance estimates
- *Predict* the number of true top-*n*% performance contained in this subset
- Check the prediction against the true performances.

The reason this method of ordinal optimization works is based on two tenets[3] :

- "Is A greater or less than B?" is a much easier question to answer than "By how much is A is greater or less than B?"—*concentrate on order and separate the "good" from the "bad" first. It is a much easier and statistically quantifiable problem.*
- Similarly "Separate the good enough from the rest" is an easier goal than "find the best"— *by accepting a softening of the goal, we can get away with using a crude model or very short simulation to narrow down the search region.*

The actual demonstration consisted of using two different models of a manufacturing cell engaged in the making of jet engine hubs.

Complex Manufacturing Model:

- Making 4 different turbine blade hub assemblies for jet engines
- Each hub requires 7-8 different machining and assembling operations in specified sequence
- Equipment for these operations separate into 8 groups to be shared by the different hubs
- Four skilled labor groups attend to these equipment
- An operation can be performed only if labor and equipment are simultaneously available, which implies potentially long queueing delays
- Lot size is a design parameter due to the tradeoff between setup and queueing delay

We evaluated 100 designs of this complex model ahead of time as a reference in our comparison with the simplified model presented next.

Simplified Manufacturing Model:

- Use only two hubs — original hub #1 and aggregate #2-#4
- Use only 4 equipment groups and two labor groups (aggregated)
- Estimate manufacturing lead time and inventory as a function of lot size for hub#1 using this simple model
- Predict the true good lot sizes (as determined by the complex model) using only the simple model

The idea is to use the complex model to represent the true system performance which under normal circumstances is unknown to us. We then use the simplified crude model and the theory of ordinal optimization to predict how the simple model can be used to narrow down quickly those designs which will (with high probability) include the actual good designs. Since, in this case, the true performance can be checked with the complex model, we can validate the effectiveness of the approach for speeding up simulation. **Figures 16-18** capture schematically what we did. The two different models are first shown in **Fig. 16.**

[3] More detail explanation can be found in the Phase II proposal being submitted concurrently and in references [12] and [16] listed therein.

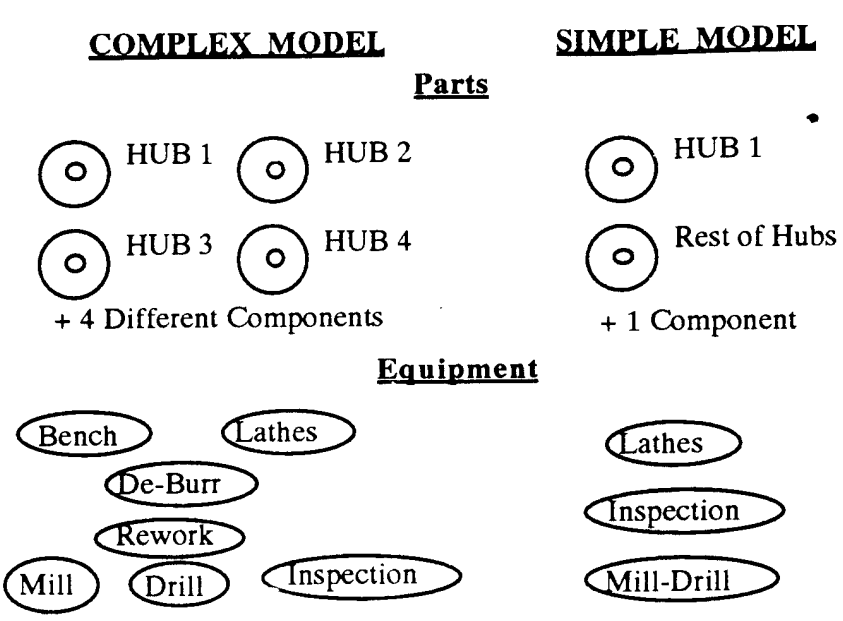

Fig. 16. The two different manufacturing models

The operation of the demonstration is explained by **Figures 17-18**.

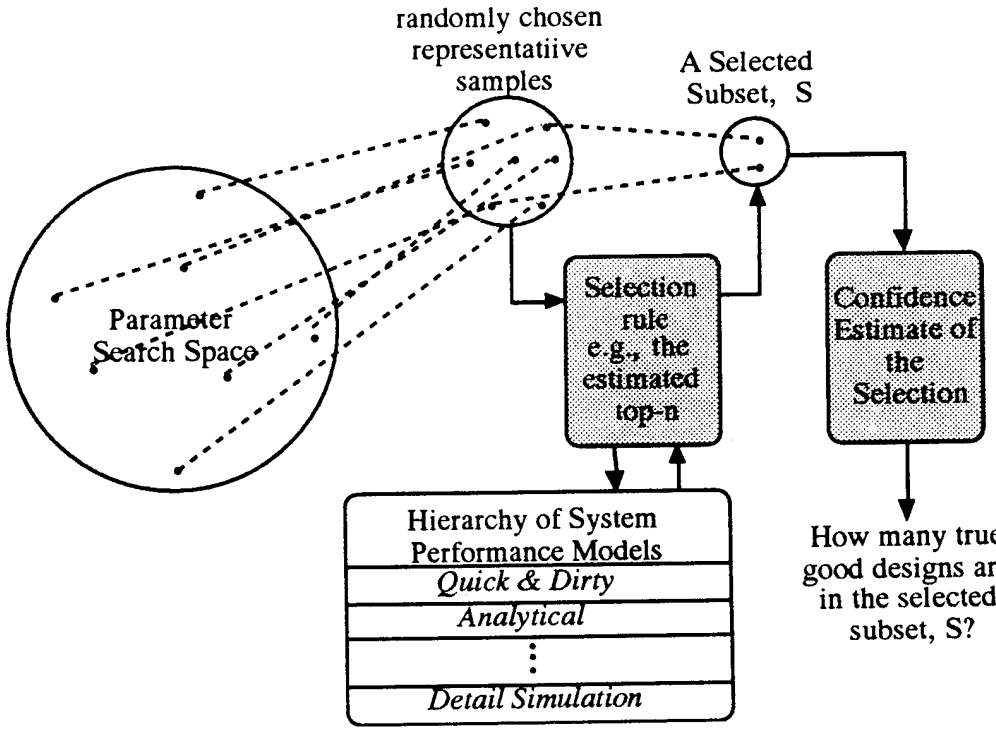

Fig. 17. Demonstration of Using a Crude and Fast Model to Predict True Performance

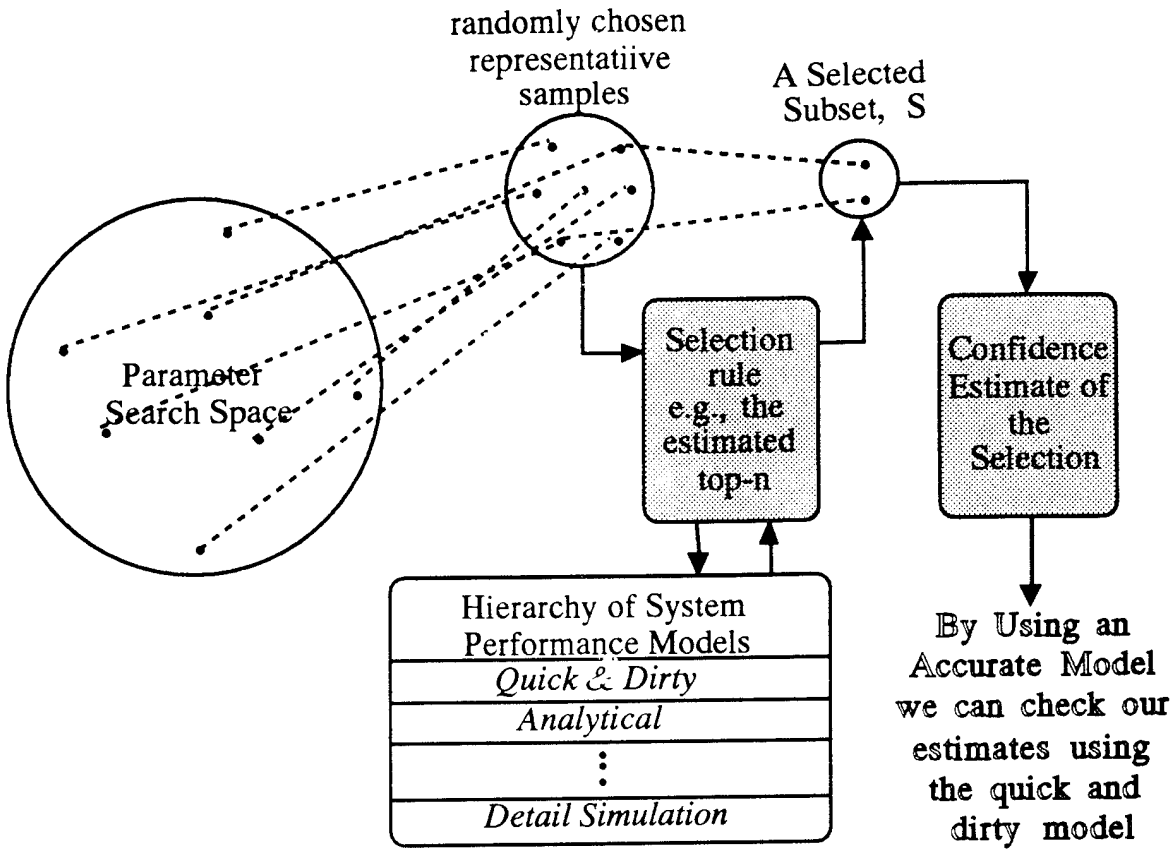

Fig. 18. Validation of the Ordinal Optimization process

The demonstration also represented an example step of the implementation of a simulation query language. To give a flavor of what we have in mind, we provide below a reasonably detailed description of this demonstration which we called a Soft Optimization Shell (SOS). It is "soft optimization" because we do not insist on getting the "best" but only the "good enough". In fact it is this tradeoff that enables us to get high values for P. It is a shell because we can plug in what ever computer model for the system under investigation ranging from back-of-the-envelope formula to complex simulation programs.

II.3.1. User Interface

The user interface is based on version 3.1 of Windows and was written using an Object-oriented language (Visual C++ v 1.5 with MFC version 2). This allowed us to focus on the details of the inter-program connections and use some commercially available tools to perform some of the tedious tasks involved. This interface can be enhanced, but it currently provides the starting point for a more visual and stimulating interaction with the user.

II.3.2. Inter-Program Interface

This tool can be thought of as an interface that allows a person who is not the author of a simulation to access and run the model. The information about a problem or model that a user requires is much less than an author needs. Also we hide the implementation language and details from the user. The point is to hide the complexity and present the user with a simple menu of input parameter choices. See **Fig. 19** below. In this dialog box, the user specifies which model is desired. In the future it will be possible with an INTERNET connection to access models that are not on the local machine but that are available through the net.

Model Name	Description
F(x) = X+ r.v.	Simple Equation
F(x) = 1 - Cos(110*p*x) + r.v.	A function with lots of local maximums
GT-HUBS	A model of Jet Engine Hub Group Technology Production Cell
GT-HUBS (Simple)	A Simplified model of GT-HUBS

Fig. 19. Available Models

II.3.3. Model Specification Files

Every model that is accessible from SOS must provide two files that describe the input data and the available output measures or results. The first file lists the inputs the user can specify, i.e. the name or title to be displayed to the user, the internal name for the model and the minimum and maximum value that the parameter is allowed to have. The second file lists the outputs the user can request to be measured including the name or title to be displayed and the internal name for the model itself. Both files use a name or title field so that different users can be shown different interfaces or one can use different languages without changing the model or its internals.

The user chooses which input parameters are to be varied in the experiment and what range is to be explored. For each input parameter, the user specifies both the minimum and the maximum values to be considered; the outer limits of which have been given by the model server in the list of possible inputs. See **Fig. 20** below for an example.

Name / Title	Field	Min	Max
No. of Lathes	Equipment.Lathe.no_available	1	None
Lathe Overtime %	Equipment.Lathe.overtime	0	30
Product 1 Volume	Part.1.demand	0	None
Product 1 Lot size	Part.1.Lotsize	1	None

Fig. 20. Input parameter list from model server

The user also chooses the output measures reported. Again the data file contains a field that gives the name or title displayed to the user and the name of the internal parameter to be measured. An example of this data file is given in **Fig. 21** below. Currently the choices are made from a long scrolling list, but it is possible to allow for a hierarchy of choices so that the user does not have to page through a long list to find 1 or 2 parameters.

Name / Title	Field
Lathe Utilization	Equipment.Lathe.total_utilization
Lathe WIP	Equipment.Lathe.total_WIP
Product 1 Lead Time	Part.1.good_flow_time
Product 1 WIP	Part.1.WIP

Fig. 21. Output measure list from model server

These files can be extended to account for a wide range of possible situations. For example we can add input fields that define whether the value must be real, integer or can be a member of a discrete set (e.g., color or sex). The user can fix input parameters by setting the range of allowable values to be only one number. The allowable outputs can be the long run average or a user could specify that values during the simulation should be returned as well. The user can request variances in addition to means or even a full distribution.

II.3.4. SOS Actions

Once the user specifies the input parameters, the output measures and the number of samples to be taken within the prescribed space, SOS generates the appropriate values for each of the samples to be run. These values and the output measure choices are then written to disk files and then sent to the model server for action. In the next section we describe the data files.

II.3.5. SOS Experiment Data Fields for Model Server

The SOS tool provides a corresponding set of files to the model server. The first file contains the input parameters and the specific values for each experiment. The second contains the requested output measures to be returned (see **Fig's 22-23**).

II.3.6. Model Server Action

The model server is activated, receives the files from the SOS tool and then runs the samples. Currently this is done on the same machine but it could be done using a parallel computer or a collection of computers available via a LAN or INTERNET. The details of this exploration are dealt with elsewhere in the report. Once the results of the samples become available, the model server returns a file to the SOS program with the output measures for each sample. These are in a simple format listing the sample number, the measure name and the value (see **Fig. 24**).

Name	Field	Value
SAMPLE 1		
No. of Lathes	Equipment.Lathe.no_available	2
Product 1 Volume	Part.1.demand	1000
SAMPLE 2		
No. of Lathes	Equipment.Lathe.no_available	2
Product 1 Volume	Part.1.demand	1250
SAMPLE 100		
No. of Lathes	Equipment.Lathe.no_available	5
Product 1 Volume	Part.1.demand	4750

Fig. 22. Input value list from SOS

Name	Field
Lathe Utilization	Equipment.Lathe.total_utilization
Product 1 Lead Time	Part.1.good_flow_time

Fig. 23. Output measure list from SOS

Name	Field	Value
SAMPLE 1		
Lathe Utilization	Equipment.Lathe.total_utilization	75
Product 1 Lead Time	Part.1.good_flow_time	45
SAMPLE 2		
Lathe Utilization	Equipment.Lathe.total_utilization	95
Product 1 Lead Time	Part.1.good_flow_time	80
SAMPLE 100		
Lathe Utilization	Equipment.Lathe.total_utilization	65
Product 1 Lead Time	Part.1.good_flow_time	23

Fig. 24. Output measure values from model server list from SOS

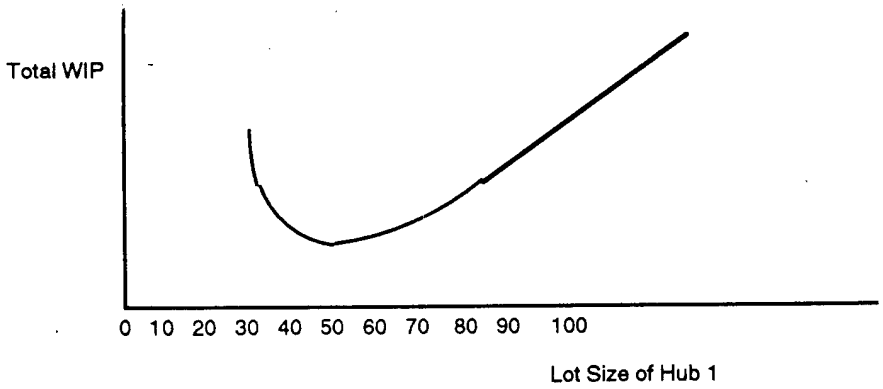

Fig. 25. Screen from SOS showing the relationship between lot size and total WIP

The results from the experiment are then displayed in both tabular and graphical format. The results can be moved to the Windows clipboard and moved to a spreadsheet. The user can sort the samples in ascending or descending order based upon any of the input or output measures. This allows the user to focus on any of the input parameters and their relationship to the output measures. In the screen shown in **Fig. 25** (taken from the GT-HUBS example) it is clear there is a relationship between the total WIP in the factory and the lot size of a major product.

II.3.7. Estimating Number of "Truly Good" Designs in Selected Set of Samples

The SOS program is capable of providing an estimate of the number of "truly good" designs that are included within an experiment. This probability is dependent upon the number of samples that the user requested, the quality of the model and the criteria that determine "truly good".

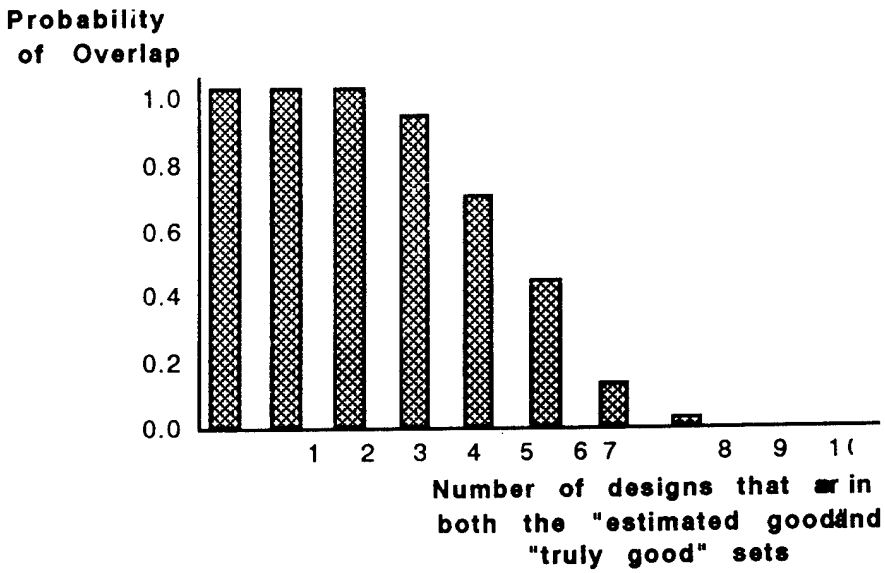

Fig. 26. Probability of overlap between the "Truly Good" designs and the "Suggested Best Designs" from the experiment

First the user has specified the number of samples he/she wants to be run. Second, the user is asked to provide a qualitative estimate of the accuracy of the model to real life. If the model is poor (i.e. it does not incorporate some important features) or the simulation has been run for a short period, the user may rate the accuracy as poor. On the other hand if the model includes many details and has been run for a sufficient length of time the accuracy is probably quite good. The third factor required is the percentage of designs that the user will call "good enough". For example, the user may state that any design that is in the top 10% will be considered good enough. Once these three data values are specified, SOS provides an estimate of how many of the top 10%

samples from the experiment will be in the true top 10% of all possible samples. An example of the probabilities that are being reported is given in **Fig. 26**.

To clarify what is being stated, we illustrate using the GT-HUBS simple model as an example. In an experiment we asked for 100 samples to be run. Within each sample we varied just the lot size of one of the parts being made. We also rated the accuracy of the model to be good and we stated that the top 10% of the designs are "good enough". From this, SOS calculated that the chances of 4 or more of the top 10 samples being in the true top 10% of all designs was 95% and that chances are 50-50 that 6 or more of top 10 samples are in the true top 10% of all designs. If we rated the accuracy to be poor, the probability that 1 or more of the 10 best sample designs is in the '"good enough" set is 90% and that it is only 50-50 that 3 or more are "good enough". In order to boost our chances of having a good design we will need to 1) increase the number of samples, 2) do something to increase our confidence in the accuracy, such as run each sample for a longer time period or change the model or 3) relax our requirements on the "good enough" set and consider the top 15% as being good enough.

II.3.8. Overview of Messages, Files, and Control

In **Fig. 27** we outline the data files and messages being passed between SOS and the Model Server and the sequence of actions being taken.

The files that flow between SOS and the model server are not based upon any detailed knowledge of the underlying model or its implementation. They are simple enough to be embedded within a more complex communication framework such as KQML or a knowledge interchange format. The value of this approach is that any complex model can have a simple front end for the user. The complexity and the accessibility of the model rest upon the model builder's choice of what input parameters and output measures are available to the user and how those choices are presented.

II.3.9. Overview of Messages, Files, and Control

We have developed a simple to use interface for a wide range of possible simulation programs. We have labeled this prototype the Soft Optimization Shell (SOS).

The interface allows a user to create projects consisting of multiple experiments. Within each experiment, the user chooses a specific model, a set of input parameters to be investigated, the range for each input parameter, the number of samples to be taken and the output measures of interest.

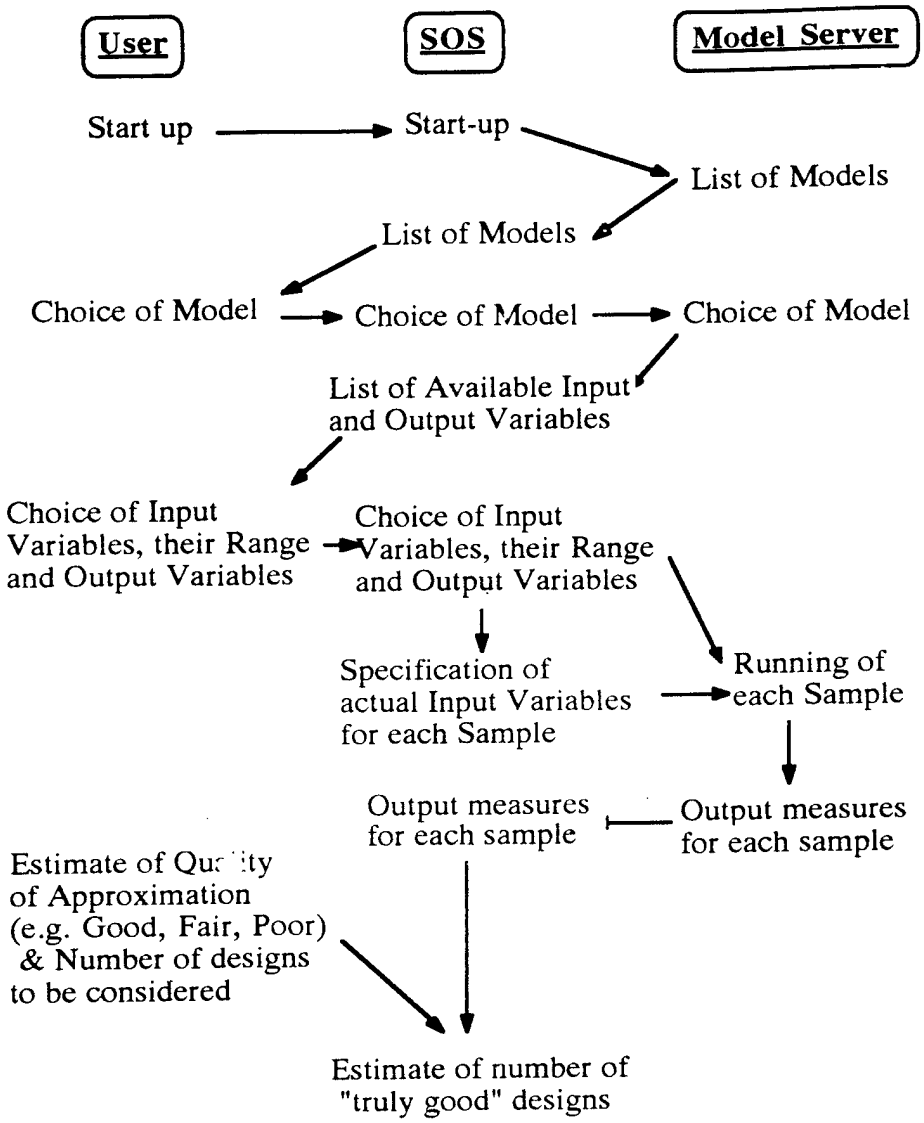

Fig. 27. Data files, messages and sequence of actions between user, SOS and model server

This tool chooses the input parameters randomly for each sample and then runs the samples. The outputs from each of the samples is recorded and the user can then manipulate the results.

The program also provides a probability estimate of the number of "estimated good" designs that are actually in the set of "truly good" designs.

III. CONCLUSIONS AND PROPOSAL FOR FUTURE RESEARCH

Based on the research progress and feasibility demonstration in this Phase I effort we conclude:

1. Orders of magnitude speed up of discrete event simulation for the purpose of performance evaluation and optimization of complex systems are possible, especially if parallel processing capabilities are present.
2. A high-level language for discrete event simulation will usher in a new era for simulation leading to increased usability among practitioners and interoperability with other high-level languages such as KQML, etc.
3. Commercial success and market potential of the end product envisioned in this research definitely exist and shall be exploited in the proposed phase II and follow-on effort.

Further details on how these conclusions lead to our proposed future research can be found in the Phase II proposal.